Modeling and Configuring UML-based Software Product Lines with SMartyModeling

Leandro F. Silva[1], **Edson OliveiraJr**[1]

[1]Informatics Department, State University of Maringá (UEM)
Maringá, Brazil.

`leandroflores7@gmail.com, edson@din.uem.br`

Abstract. *Variability modeling in UML-based Software Product Lines (SPL) has been carried out mostly using the UML Profiling mechanism. However, there is no UML-based SPL life cycle supporting tool, which takes advantages of UML standard diagrams in a controlled environment exclusively for it. In this scenario, we developed SMartyModeling, which allows SPL modeling on UML models, use of different visualization techniques to SPL/variability information, traceability, and configuration of products. The architecture of SMarty-Modeling was instantiated based on VMTools-RA, a Reference Architecture for software variability tools. This paper presents the SMartyModeling in an architectural viewpoint, describes its requirements, views, and elements selected from VMTools-RA and the decisions made during the instantiation process. We also present examples of using the environment, modeling an adaptation of the Mobile Media SPL and generating a product. We also discuss lessons learned and performed evaluations.*

1. Introduction

The requirement to achieve high levels of quality in software products has concentrated efforts from academia and industry, demanding quality in the development process and in the products created. Software engineering has evolved to create and improve methods for software development in a faster and more quality way, allowing to reach the established deadlines and goals [Long et al. 2017].

The reuse of requirements, architectures and other artifacts in a high level of abstraction is efficient in software development traditional techniques, focused on the source code. In this context, the Software Product Line (SPL) approach has been consolidated as a technique for systematic reuse in many companies around the world [Linden et al. 2007]. The SPL approach comprises a set of essential activities such as variability management, which is a key issue for success in adopting SPL. The adoption of the SPL approach aims at increasing the reuse of requirements and artifacts, thus reusing documents, source code and artifacts and ensuring better quality control to software production in large-scale [de Almeida 2019].

The industry has increasingly required the support of tools for the SPL approach [Meinicke et al. 2014]. It has been developing its own solutions to support the SPL life cycle. However, the current support tools are mainly restricted to the problem space based on feature modeling [Bashroush et al. 2017], such as FeatureIDE, AspectJ, AHEAD, Captain Feature, and DOPLER [K. U. and Nandhini 2017]. To provide support to the solution space in SPL, we developed SMartyModeling, an environment for SPL modeling, with support to UML stereotype-based approaches such as PLUS [Gomaa 2006], Ziadi [Ziadi et al. 2003], and SMarty [OliveiraJr et al. 2010]. SMartyModeling was built based on the architectural requirements, views and elements defined by VMTools-RA [Allian 2016], a Reference Architecture (RA) for software variability tools.

2. Background and Related Work

2.1. Software Product Lines and Variability Management

The SPL approach has been consolidated as a technique for systematic reuse in many companies around the world [SPLC 2018]. SPL engineering has proven to be the methodology for developing a diversity of software products and software-intensive systems at lower costs, in shorter time, and with higher quality [de Almeida 2019]. SPL is based on the principle of systematic reuse of requirements and artifacts, including documents, source code and artifacts, ensuring better quality control to software production in large-scale. SPL corresponds to a set of software systems that share common and manageable features that contain the needs of a particular segment or mission [Linden et al. 2007].

SPL has consolidated Variability Management (VM) as one of its essential activities for a successful non-opportunistic reuse. VM encompasses a number of activities, such as identification and modeling of system variants, implementation (realization), and selection and configuration (product derivation). Software variability represents the product features in terms of variants and variation points [Capilla et al. 2013]. Many different approaches to variability representation have been discussed over the years. Thus, variability can be formally defined by four fundamental elements [Linden et al. 2007]:

- **Variation Point:** describes where differences exist in the SPL artifacts;
- **Variant:** the different possibilities that exist to satisfy a variation point;
- **Variability Dependencies:** this is used as a basis to denote the different variants that are possible to fill a variation point. The notation includes a cardinality which determines how many variants can be selected simultaneously; and
- **Constraint Dependencies:** they describe dependencies among certain variant selections. There are two forms:
  - *Requires*: the selection of a specific variant may require the selection of another variant (perhaps for a different variation point).
  - *Excludes*: the selection of a specific variant may prohibit the selection of another variant (perhaps for a different variation point).

2.2. The SMarty Approach for UML-based SPLs

SMarty-based Management of Variability (SMarty) is a VM approach based on UML stereotypes. SMarty consists of an UML profile, named *SMartyProfile*, which represents variability through stereotypes and tagged values, and a process, named *SMartyProcess*, which guides the user in identification, representation and tracking of variabilities in SPL models based on UML [OliveiraJr et al. 2010].

SMarty supports VM in use case, class, activity, component and sequence diagrams. The SMartyProfile is based on the inter-relationship of the main concepts of SPL, as Variabilities, Variation Points and Variants (Section 2.1). SMartyProfile defines the following stereotypes [OliveiraJr et al. 2010]:

- **variability:** represents variabilities in an UML note. It has the following attributes: name: name used to refer to a variability; minSelection: minimum number of variants selected to solve a variation point or variability; maxSelection: maximum number of variants selected to solve a variation point or variability; bindingTime: moment of variability resolution; allowsAddingVar: indicates whether new variants can be included after resolution of a variability; variants: collection of instances associated with variability; realizes: collection of variability of lower level models that realizes variability.
- **variationPoint:** stereotype of variation point;
- **mandatory:** represents this variant must necessarily be present in any product;
- **optional:** represents an optional variant;
- **alternative_OR:** indicates the existence of a group of inclusive variants. Different combinations of inclusive variants can be selected for the resolution of a variation point or variability;
- **alternative_XOR:** indicates the existence of a group of exclusive variants. Only one variant of this group can be selected for the resolution of a variation point or variability;
- **mutex:** represents the mutually exclusive relationship between variants;
- **requires:** represents a complement relationship between two variants.

The full description of the SMartyProfile and application examples are available in [OliveiraJr et al. 2010].

2.3. VMTools-RA: a Reference Architecture for Software Variability Tools

VMTools-RA stands for Variability Management Tools - Reference Architecture. Reference Architectures (RA) are considered a predefined standard designed for a specific business context [Nakagawa et al. 2014]. In this case, VMTools-RA encompasses knowledge and practice for developing and evolving variability tools [Allian 2016].

VMTools was created from information organized from three sources: applicable standards in the context of VM, a systematic mapping study on software variability tools, and secondary studies on software variability tools. The information analyzed served as basis for establishing 21 architectural requirements for VMTools-RA. Requirements identified for VMTools-RA were divided into two groups: (i) nine architectural requirements related to variability management implementation and (ii) 12 architectural requirements addressed to organizational support and market analysis for the maintenance of quality and evolution of variability management [Allian 2016].

VMTools-RA defines the Module View, represented in an UML class diagram, responsible for describing specific functionalities through modules (packages), data flow, and interface. VMTools-RA consists of four modules encapsulate functionalities [Allian 2016]:

- Support: provides technologies for storage of system data (e.g., information, models, and domain assets) and importing and exporting information;
- Organizational Management: is responsible for the organizational process for maintaining quality of software products. It supports an environment of communication and sharing of variability management information through different stakeholders;

- Domain Analysis: is responsible for domain asset acquisition and management and is designed with a middleware for integration to different domain analyses and requirement tools; and
- Variability Management: involves four sub-modules: Variability Modeling: identifies and models variations, traceability, documentation and composition rules related to constraints dependencies; Variability Validation: validates variability models by consistency check using arithmetical checkers or logic solvers; Variability Decision: realizes variability in different binding times; and Variability Evolution: manages variability evolution.

Figure 1 presents the Module View of VMTools-RA. Figure 1 presents the modules described, including their respective sub-modules and associations. The modules are presented emphasizing the flow of information and how the modules communicate.

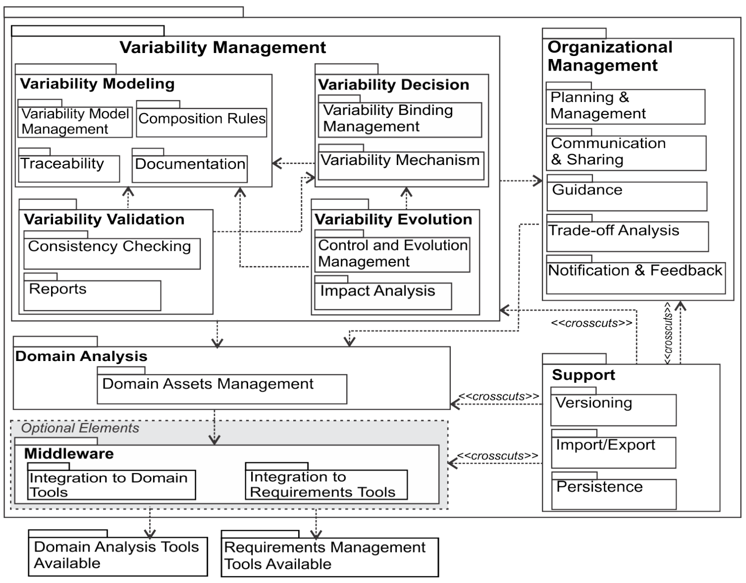


Figure 1. Module View of VMTools-RA [Allian 2016].

3. SMartyModeling: an Environment for UML-based SPLs

SMartyModeling[1] is an environment for engineering UML-based SPLs in which variabilities are modeled as stereotypes using any compliant UML profile. We are currently adopting the SMarty approach (Section 2.2) for VM. The environment supports use case, class, component, sequence, and activity diagrams. It has as main features in its current version: variability modeling and constraining, matrix-based support to traceability among SPL elements, and specific product configuration.

The creation of a RA as VMTools-RA (Section 2.3), intended for software variability tools, allows the use of consistent models, including efficient and tested solutions to reduce the complexity for the development of a new environment for variability on SPL context. In this scenario, the SMartyModeling architecture was designed and instantiated from VMTools-RA.

[1]SMartyModeling is available at `https://github.com/leandrofloress/demo_SMartyModeling_tool` for free using according to Creative Commons 4.0 licenses.

We designed the SMartyModeling architecture according to the VMTools-RA description of elements and views. As VMTools-RA is widely designed for software variability, we decided to develop an environment for SPL variability. Thus, the first step was to select the architectural requirements described by VMTools-RA for the construction of the new environment. We consider the following architectural requirements: Manage domain assets (models, UML diagrams, features) [AR.1.1]; Build variability models [AR.1.2]; Consider composition rules (e.g., cardinalities and dependencies) [AR.1.3]; Document variability models in detail [AR.1.5]; Validate conformance among variability models by using automatic consistency check [AR.1.6]; Implement impact analysis to determine what changes are required [AR.2.2]; Support import/export of variability assets and relevant information [AR.2.9]; and Offer an efficiency scalability of feature models [AR.2.11]. We adapted certain VMTools-RA views and elements for the context of SPL specifically aiming at: identifying, constraining, representing, and tracing variabilities on SPL context. Figure 2 presents the Module View of VMTools-RA instantiated for SMartyModeling architecture.

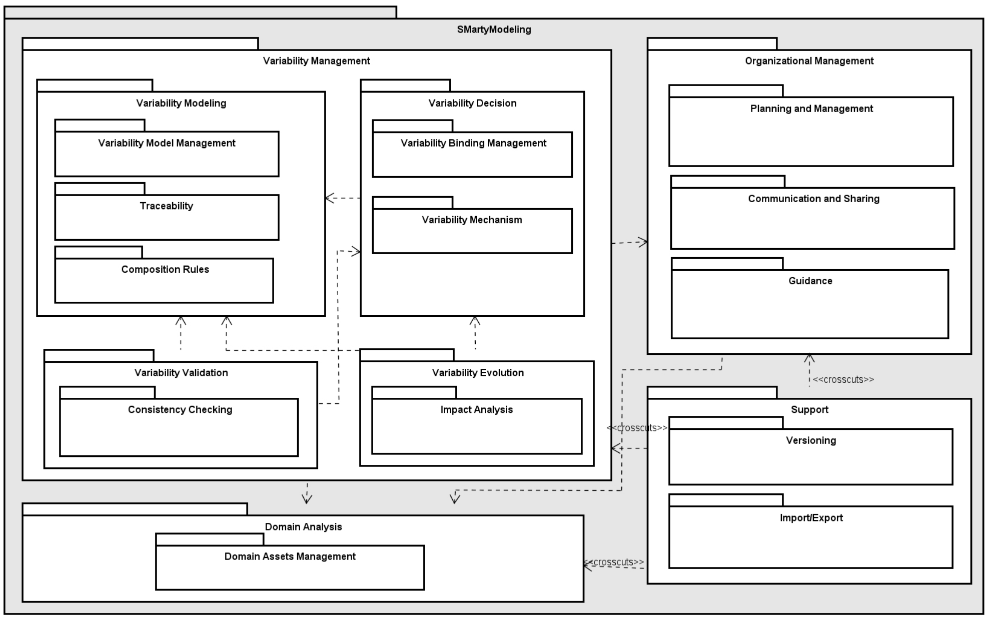


Figure 2. Module View Instance for SMartyModeling.

3.1. Why do We Need SMartyModeling?

The industry has increasingly required the support of tools for the SPL approach. However, the current support tools are mainly restricted to the problem space based on feature modeling [Bashroush et al. 2017]. In this scenario, we developed SMartyModeling, an environment for SPL modeling, with support to most UML stereotype-based approaches. The main benefit of SMartyModeling is provide support for the main activities related to SPL Management. SMartyModeling provides users control over the data and the integration of new functionalities with existing tools.

3.2. SMartyModeling Component-based Architecture

VMTools-RA does not specify an specific architectural pattern or style. Therefore, we built SMartyModeling under the Model-View-Controller (MVC) pattern. We used Java

SE for desktop. From the information considered from VMTools-RA, such as requirements, elements and modules, the functionalities were organized, designing the solutions proposed in classes and separating packages and classes according to the modules. SMartyModeling architecture is composed of classes and interfaces organized in four main packages: `Model`, `Controller`, `View`, and `File`. The latter has two sub packages: `Export` and `Import`. Package `Model` comprises the main classes and interfaces of the environment: `Project`, which is related to several other essential classes and interfaces, such as: `Profile`, `Traceability`, *`Diagram`*, `Stereotype`, and `Product`. Figure 3 depicts the internal organization of the `model` package aiming at the separation of concerns, increasing cohesion, and decreasing coupling.

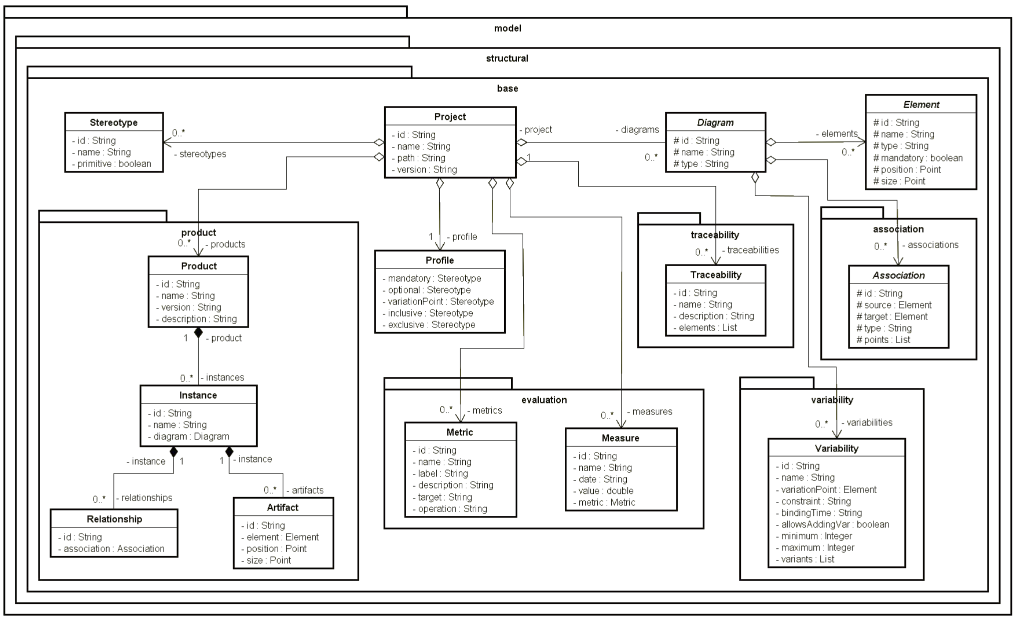


Figure 3. Classes and Interfaces of the Package `model`.

The class `Project` contains a reference to a set of *`Diagram`*, which allows variability modeling in use case, class, component, sequence, and activity diagrams. Each *`Diagram`* is a composition of *`Element`*, *`Association`*, and `Variability`. We defined the structure of a `Variability` taking into consideration tagged-values of a variability definition in the SMarty approach. Therefore, a `Variability` is related to a variation point and a set of variants to resolve such variation point.

The class `Project` is related to a `Profile`, which defines the role of each `Stereotype` for modeling variabilities. For instance, user might use the Gomaa's profile [Gomaa 2006], the Ziadi et al.'s [Ziadi et al. 2003], or the SMartyProfile [OliveiraJr et al. 2010]. We set the latter as default. One or more SPL specific `Product` might be configured for a `Project`. Each `Product` is composed of `Instance`, which is a composition of `Relationship` and `Artifact`. An `Instance` class refers to a `Diagram`, thus a `Product` is composed of a set of `Diagram`.

3.3. SMartyModeling Current Features

SMartyModeling can be divided into features. For each feature, architectural decisions were made taking into account the description of the requirements presented by VMTools-

RA. Therefore, currently, SMartyModeling consists of four main features:

- **Modeling of Diagrams:** feature responsible for defining the classes responsible for allowing the modeling of UML diagrams, including their elements and associations. The diagrams supported by the environment are: features, use case, class, component, activity and sequence;
- **Elements Traceability:** feature responsible for allowing the traceability of the elements modeled in different diagrams in the environment, allowing to associate a specific interest to a description and a set of elements;
- **Variability Management:** feature responsible for allowing variability management, with the class structure defined according to the description of the SPL concepts. By default, the environment uses the stereotypes described by the SMarty approach (Section 2.2); and
- **Product Instantiation:** feature responsible for guiding the user in the instantiation process, respecting the selection of optional elements, resolving the variability constraints and configuring the products;

3.4. How was SMartyModeling Evaluated?

Initially, we analyzed the feasibility of SMartyModeling in two studies: one qualitative and one experiment. All instrumentation and data are available at `https://zenodo.org/record/3336227`. The qualitative study allowed identifying the points to be improved, in particular, the limitation of the interface in manipulating the elements and defining the associations of SMartyModeling elements. We then, raised hypotheses mainly in relation to the extent SMartyModeling improves the application of UML-based SPL concepts and how limitations in the interface interfere SPL modeling.

Then, we performed an experiment to identify efficiency and effectiveness of SMartyModeling and Astah, and to provide initial evidence on the feasibility of SMartyModeling and its further development. Overall results support higher efficiency of Astah in relation to SMartyModeling. On the other hand, SMartyModeling had advantage in effectiveness for the sample of invited participants, based on a hypothesis test and discussed threats to validity.

4. Modeling SPLs and Configuring SPL Products

To exemplify SPL modeling with SMartyModeling, we chose Mobile Media (MM). MM is an SPL that implements mobile applications to manipulate media (photos, music and video) on mobile devices. In our example, we adapted a MM use case model. Figure 4 presents the adaptation made for the MM use case model. `User` actor and the `Delete Album`, `View Album` and `Add Album` use cases are mandatory, while the `Receive Photo` use case is optional. Two variabilities are defined, both having the `View Album` use case as a variation point. The first variability is inclusive and has four variants: `Add Photo`, `Delete Photo`, `View Photo` and `Play Music`. The second variability is exclusive and has two variants: `Set Favorite` and `View Favorites`.

The next step is to create a new Product. As presented in Figure 3, an specific product is composed of a set of Instances, directly associated with a specific Diagram. SMartyModeling guides the instantiation process of a Diagram, in a process consisting of three stages:

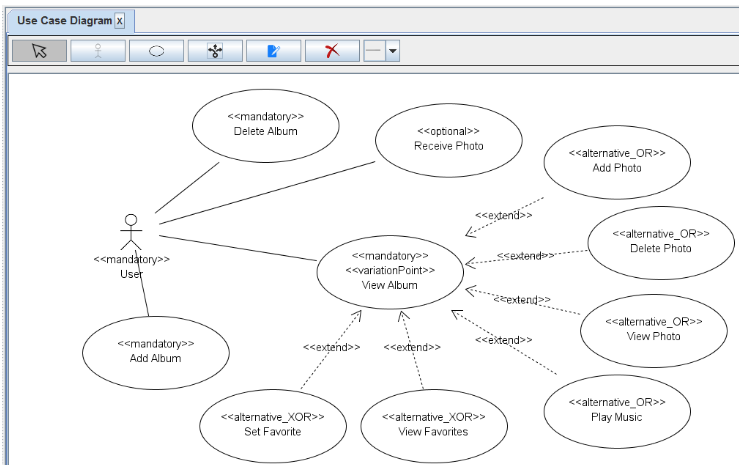

Use Case Diagram
<<mandatory>> Delete Album
<<optional>> Receive Photo
<<alternative_OR>> Add Photo
<<extend>>
<<alternative_OR>> Delete Photo
<<extend>>
<<mandatory>> User
<<mandatory>> <<variationPoint>> View Album
<<extend>>
<<alternative_OR>> View Photo
<<extend>>
<<mandatory>> Add Album
<<extend>>
<<extend>>
<<alternative_OR>> Play Music
<<alternative_XOR>> Set Favorite
<<alternative_XOR>> View Favorites

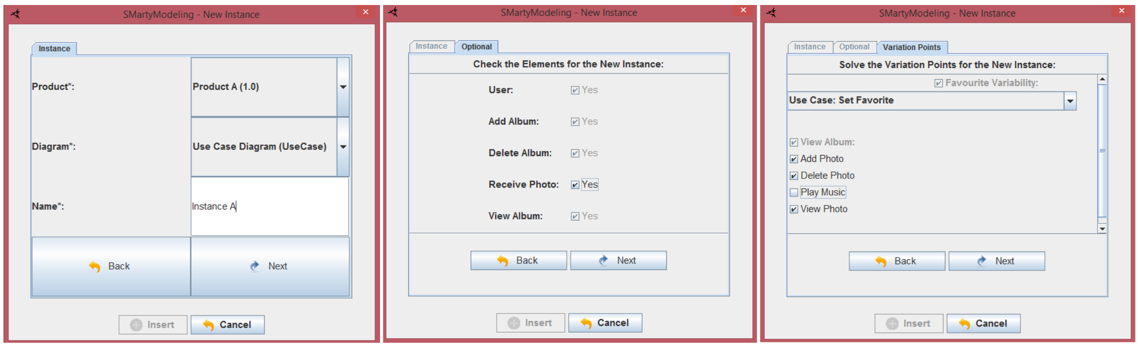
Figure 4. Mobile Media Use Case Diagram (Adapted).

- **Initial Information:** an initial panel is presented to the user, requesting information about which Product the Instance will be part of and selecting the Diagram that will be used as the basis for instantiation;
- **Selection of Optional Elements:** a panel is presented with all the elements of the diagram, with a checkbox associated with each element, and the mandatory elements of the diagram are fixed as marked and the optional elements are marked according to the user's option;
- **Resolution of Variability:** a panel is presented, according to the mandatory and optional elements selected in the previous step, listing the elements that are marked as variation point. Thus, for the diagram's variability, the following are presented:
  - **Inclusive Variability:** with the name of Variability, followed by its respective set of variants, associating each variant with a checkbox, thus allowing the user to resolve the variability, choosing the variants for the new Instance; and
  - **Exclusive Variability:** with the name of Variability, followed by a combobox with the variants defined for the variability, thus allowing the user to solve the variability, choosing a single variant for the new Instance.

Following the example, the Figure 5 shows the sequence of steps for instantiating the diagram shown in Figure 4. In the first panel of the Figure 5, we selected the base Diagram for instantiation and the Instance name. The second panel presents the mandatory elements and we selected the optional `Receive Photo` use case. And in the third panel, we solved the exclusive variability, choosing the `Set Favorite` use case and we solved the inclusive variability, selecting the `Add Photo`, `Delete Photo` and `View Photo` use cases as variants.

5. Lessons Learned on Developing SMartyModeling

The development of the SMartyModeling environment involved a series of decisions, including the views and elements described by VMTools-RA to the planning, definition, architecture instantiation and implementation of the environment. Therefore, the first lesson



Product*: Product A (1.0)
Diagram*: Use Case Diagram (UseCase)
Name*: Instance A

Check the Elements for the New Instance:

User: ✓ Yes
Add Album: ✓ Yes
Delete Album: ✓ Yes
Receive Photo: ✓ Yes
View Album: ✓ Yes

Solve the Variation Points for the New Instance:

✓ Favourite Variability:
Use Case: Set Favorite

✓ View Album:
✓ Add Photo
✓ Delete Photo
☐ Play Music
✓ View Photo

Figure 5. Interface with the 3 stages for instantiating the Moblie Media Diagram.

was that although VMTools-RA presents a complete and safe view of the concept of variability, we need to adapt these definitions to the context of SPL. During the architecture instantiation process, it was necessary to design classes and interfaces before implementation. Therefore, we design the classes in a generic way considering the requirements and views described by VMTools-RA, however, leaving the architecture flexible for changes. As a main contribution, we can mention the instantiation and implementation of an architecture from VMTools-RA, resulting in a practical application, which is part of an maturity process of an AR. The initial evaluations (Section 3.4) indicated good results, mainly in relation to the application of the SPL concepts. As a limitation, we can consider, from the point of view of instantiation, the restriction in developing solutions for the modules related to the most complete documentation, reports, and evolution of variability, and activities of organizational management. From a practical point of view, the evaluations also indicated limitations reported by the participants, in particular, regarding the definition of associations and event handling in the modeling panel.

6. Final Remarks

We presented SMartyModeling, a tool that supports the main activities on SPL management. We described the main steps of the development process, starting from the selection of VMTools-RA architectural requirements, views and elements, the adaptation of the instantiated architecture to the SPL context, the lessons learned in the implementation and the initial evaluations performed. VMTools-RA considers software variability tools in a broader context. Therefore, the first decision was to restrict the representation of variability for the context of SPL. However, regardless of this restriction, the concepts and elements described mainly in terms of Variability Management were taken into account, being naturally adapted to the SPL domain.

The instantiation and implementation of an architecture from VMTools-RA collaborates mainly with a practical application and is part of a maturity process for the RA. Among the points to improve the environment it could include more elements described by VMTools-RA. In particular, regarding the evolution of variability, planning a solution that included a complete control with information and management of the evolution of variability during the project.

We have designed two more evaluations for SMartyModeling. The first aims to evaluate the SMartyModeling instantiation process according to the guidelines presented by VMTools-RA. For this evaluation, SPL experts will be invited, presenting the requirements, views and guidelines of VMTools-RA,as well as the documents with the instantiation decisions, architecture description and documentation of SMartyModeling. The second one aims at evaluating usability, without the objective of comparing it with another tool. Such evaluation will be carried out with potential users of SMartyModeling by extending the Technology Acceptance Model with the System Usability Scale.

References

Allian, A. P. (2016). VMTools-RA: a Reference Architecture for Software Variability Tools. Master's thesis, State University of Maringá. in portuguese.

Bashroush, R., Garba, M., Rabiser, R., Groher, I., and Botterweck, G. (2017). Case tool support for variability management in software product lines. *ACM Comput. Surv.*, 50(1):14:1–14:45.

Capilla, R., Bosch, J., and Kang, K. C. (2013). *Systems and Software Variability Management: Concepts, Tools and Experiences*. Springer.

de Almeida, E. S. (2019). Software Reuse and Product Line Engineering. In Cha S., Taylor R., K. K., editor, *Handbook of Software Engineering*, pages 321–348. Springer, Cham, Switzerland.

Gomaa, H. (2006). *Designing software product lines with uml 2.0: From use cases to pattern-based software architectures*. Springer-Verlag.

K. U., K. and Nandhini, M. (2017). Classification of Tools For Feature-Oriented Software Development A Comprehensive Review. *International Journal of Computer Sciences and Engineering*, 5:329–337.

Linden, F. J. V. D., Schmid, K., and Rommes, E. (2007). *Software product lines in action: The best industrial practice in product line engineering*, volume 20. Springer-Verlag New York, Inc.

Long, F., Amidon, P., and Rinard, M. (2017). Automatic inference of code transforms for patch generation systems. In *ACM SIGSOFT FSE*, pages 727–739.

Meinicke, J., Thum, T., Schroter, R., Benduhn, F., and Saake, G. (2014). An Overview on Analysis Tools for Software Product Lines. In *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools - Volume 2*, page 94–101, New York, NY, USA. Association for Computing Machinery.

Nakagawa, E. Y., Guessi, M., Maldonado, J. C., Feitosa, D., and Oquendo, F. (2014). Consolidating a process for the design, representation, and evaluation of reference architectures. In *WICSA*, pages 143–152, Washington, DC, USA.

OliveiraJr, E., Maldonado, J. C., and Gimenes, I. M. S. (2010). Systematic Management of Variability in UML-based Software Product Lines. *Journal of Universal Computer Science (JUCS)*, pages 2374–2393.

SPLC (2018). *Software Product Line Conference - Hall of Fame*. URL: https://splc.net/fame.html.

Ziadi, T., Hélouët, L., and Jézéquel, J.-M. (2003). Towards a uml profile for software product lines. In *PFE*, pages 129–139. Springer.